Prefetching for Visual Data Exploration*

Punit R. Doshi, Elke A. Rundensteiner and Matthew O. Ward
Computer Science Department, Worcester Polytechnic Institute, Worcester, MA 01609
{punitd,rundenst,matt}@cs.wpi.edu

Abstract

Modern computer applications, from business decision support to scientific data analysis, utilize data visualization tools to support exploratory activities. Visual exploration tools typically do not scale well when applied to huge data sets, partially because being interactive necessitates real-time responses. However, we observe that interactive visual explorations exhibit several properties that can be exploited for data access optimization, including locality of exploration, contiguous queries, and significant delays between user operations. We thus apply semantic caching of active query sets on the client side to exploit some of the above characteristics. We also introduce several prefetching strategies, each exploiting characteristics of our visual exploration environment. We have incorporated caching and prefetching strategies into XmdvTool, a public-domain tool for visual exploration of multivariate data sets. Experimental studies using synthetic as well as real user traces are conducted. Our results demonstrate that these proposed optimization techniques achieve significant performance improvements in our exploratory analysis system.

1 Introduction

Whether the domain is stock market data, scientific measurements, or the distribution of sales, visualization is becoming an increasingly popular technique for data exploration. When presented with visual depictions of the data, humans can often easily detect interesting patterns as well as outliers, which may be more difficult to identify and rate as relevant with automated techniques [22]. Interactive visual navigation tools play an important role in aiding users to find their way through such large data sets. By presenting information visually and allowing dynamic user interaction through direct manipulation paradigms, it is possible to traverse larger information spaces in a shorter time and to discern relevant knowledge effectively. Significant effort has thus been spent on developing effective methods to display and visually explore information [1, 21, 17, 10, 11].

Most of these visualization tools nowadays still execute on data that is first fetched from the file system and loaded entirely into main memory. However, as typical sizes of datasets become larger (on the order of giga-bytes or more), current datasets can no longer be held entirely in main memory. We thus must scale visual tools to work with large data sets stored on persistent storage without sacrificing the near real-time response times required to service user's navigation and exploration requests. Note that even a small movement in the user's navigation tool may mean executing a new query to retrieve the selected data, potentially resulting in a high data access rate. Being an interactive feedback-driven paradigm, it is critical that the user receives responses to her navigation requests with little or no time lag.

Client-side caching is one important technology for achieving quick response in client-server applications. Caching provides the advantage of allowing most frequently requested data to be kept in the cache to speed up the response. Hence, we put forth that managing such client-side caches is a critical component of visual exploration systems. Furthermore, as we have identified through our experience with a multivariate visual exploration tool [26], unlike in traditional high-transactional multi-user database applications, the queries coming from the visual interface tend to be contiguous and follow some pattern rather than being random. To enable excellent performance of subsequent user operations, we thus propose to apply semantic caching techniques [4, 13] for maintaining the client cache. This logically groups data in the cache and thus reduces cache lookup overhead due to the compact query-based organization of the cache content.

To further improve the performance, we have also developed an array of speculative non-pure prefetching methods. When the system is idle, a prefetcher will bring data into the cache that is likely to be used next. These strategies, working hand in hand with the semantic caching scheme, exploit the characteristics of the exploratory environment in order to optimize the contents of the cache. These characteristics include: (1) incremental refinement of queries formulated via a visual query tool in the quest to explore the details of a particularly interesting subregion of the data space, (2) contiguous queries by one user rather than unrelated adhoc queries requested by different users concurrently for unrelated purposes, and (3) predictable user behavior due to limited means of data requests via the visual interactive tools and also due to existence of some typical styles of exploration.

The proposed prefetching techniques have been incorporated into XmdvTool [8, 28, 27, 26], a public-domain tool for multivariate visual exploration developed at WPI[1]. This tool, supported by several NSF grants, aims to scale up existing visual exploration techniques to work for large data sets and for data with high dimensions. Our experiments confirm the important role of caching and prefetching in visualization applications and in particular demonstrate that the benefit of using prefetching significantly exceeds the result gained by using caching only.

The remainder of the paper is structured as follows: Section 2 identifies key characteristics of interactive visualization environments. Section 3 explains our approach to semantic caching, while Section 4 introduces our prefetching strategies. Section 5 discusses the XmdvTool system implementation, while Section 6 presents our experimental evaluation. Sections 7 and 8 present related work and conclusions, respectively.

*This work is supported in part by NSF grants IIS-9732897, IRIS 97-29878 and IIS-0119276.

[1] XmdvTool was demonstrated at ACM SIGMOD 2002 [20], and yearly releases of the software can be found in our XmdvTool homepage [26].

2 Multivariate Data Visualization

We now introduce XmdvTool, a visual exploration tool for multi-variate data [8, 26], which represents not only the driving force of this work but also the testbed into which we incorporate our proposed techniques and then evaluate them. The proposed optimization techniques are however also applicable to other exploratory analysis tools, as long as they meet the typical interactive visual exploration characteristics identified below.

XmdvTool is a public-domain software package we have been developing at Worcester Polytechnic Institute for the interactive visual exploration of multi-variate data sets [25, 7]. XmdvTool supports an active process of discovery as opposed to passive display. The major hurdles it overcomes are the problems of display clutter (too much data at once tends to confuse viewers and too many dimensions hinders the users from finding useful data features) and intuitive navigation (what tasks comprise a typical exploration process and how they can be made intuitive). Given that interactive exploration must be supported in near real-time, we designed the necessary backend support for efficient data access for the above operations scalable for large data sets.

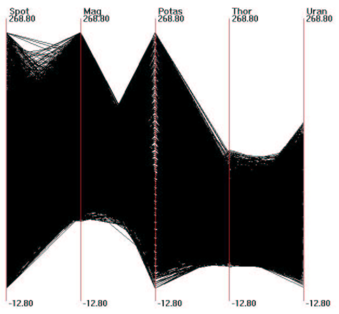

Figure 1: Flat Parallel Coordinates Display of the 5-dimensional Minerals data set with 16,384 data points.

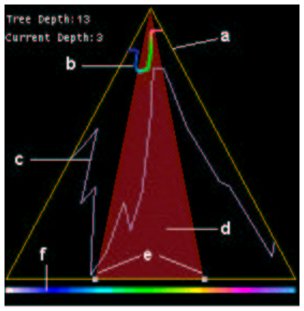

Figure 2: Structure-based brush in XmdvTool. (a) Hierarchical tree frame; (b) Contour corresponding to current level-of-detail; (c) Leaf contour approximates shape of hierarchical tree; (d) Structure-based brush; (e) Interactive brush handles; (f) Color map legend for level-of-detail contour.

XmdvTool supports four different displays, such as Scatterplots, Star Glyphs, Parallel Coordinates and Dimensional Stacking [26]. Figure 1 shows the parallel coordinates display of a five dimensional data set having 16,384 records. In this display technique, each of the N dimensions is represented as a vertical axis, and the N axes are organized as uniformly spaced lines. A data element in an N-dimensional space is mapped to a polyline that traverses across all of the N axes crossing each axis at a position proportional to its value for that dimension. For example, in Figure 1, the polyline that intersects the "Spot" axis at value 119, the "Mag" axis at 149, the "Potas" axis at 41, etc. displays the tuple (Spot, Mag, Potas, Thor, Uran) = (119, 149, 41, 56, 56).

As seen from Figure 1, displaying all the data to the user at the same time results in display clutter. Hence we need to provide user with operations such as drilling down and rolling up the level of detail of the data. Towards this end, XmdvTool first clusters the data points into a cluster hierarchy, and then associates aggregate information with the resulting clusters [25], such as extents and level-of-details to map the hierarchy into a two-dimensional plane. Different levels in the cluster tree represent different degrees of abstraction of the data.

In order to improve the support for visual navigation through this cluster tree of data sets with millions or more records, we have designed a visual navigation tool that we term *structure-based brush* (see Figure 2) [7]. Structure-based brush can be used to explore the data by interactively selecting and displaying the data at different levels of detail of the cluster hierarchy. In Figure 2, the brushing tool component marked '*e*' selects cluster(s) to be displayed, while '*b*' selects the level of detail for the selected cluster(s). While exploring the data, a user may navigate by sliding the extents of '*e*' horizontally to select a particular cluster in the tree hierarchy, or by moving the level brush '*b*' vertically to display data at different levels of detail.

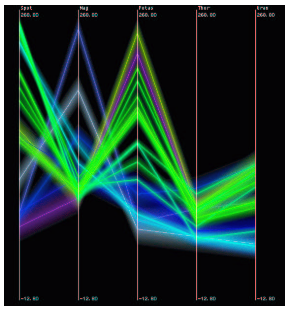

Figure 3: Hierarchical Parallel Coordinates Display of Minerals data set after Drill-down operation.

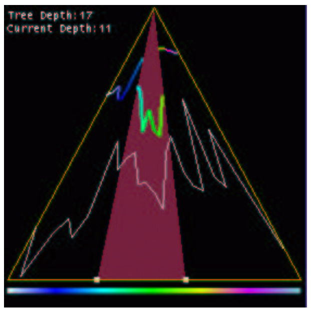

Figure 4: Drill-down operation of the Structure-based brush.

Figure 3 shows the display of the same data set now as the user performs a drill-down operation using the structure-based brush; corresponding to sliding the level-of-detail 'b' to the setting of the brushing tool shown in Figure 4.

The user navigation operations expressed by our brushing tool are translated into queries to the database. The queries are *contiguous* rather than ad-hoc, since the visual interface provides controlled means of expressing navigational requests via the structure-based brush. Such contiguity of user queries can be exploited when caching queries, as there is a high probability of a partial query result from a prior query still being relevant (and thus in our cache) for the next user request.

Second, we note that user may be viewing the data around a particular region for a while before moving to another region. In other words, the user navigation tends to be composed of several small and *local* movements of the brushing tool rather than major global and unrelated movements (queries).

Furthermore, users' exploratory movements are somewhat more *predictable* when they explore the data using such visualization tools, as such explorations are different from, say, random accesses via an ad-hoc SQL query interface. We thus postulate that prefetching may be a suitable mechanism for improving the performance of such exploratory applications.

Since the user will be examining the visual displays for interesting patterns, there typically would be *delays* between two user operations. Such delays could provide us with the opportunity to prefetch highly probable data into main memory during idle times.

To summarize, typical characteristics of visual exploration tools that can be exploited for caching and prefetching are: (1) contiguous queries passed to the database, (2) locality of exploration and thus data access, (3) predictable user's exploratory movements, and (4) significant delays between user operations.

3 Semantic Caching for XMDVTool

Semantic caching [13, 4] is a popular strategy for providing efficient support for access to cached data. Traditional caching schemes [5] either cache pages of data (even when possibly only a small subset of the objects on the disk page may be required) or individual objects (that requires heavy lookup overhead at the granularity of each object in the cache). In contrast, semantic caching caches dynamic groups of objects that logically belong together and thus can be described by one semantic descriptor, such as a query expression indicating the logical conditions that a given group of cached objects meets.

Semantic caching works well in environments that exhibit primarily queries that are contiguous in nature, which is typical for visual exploration environments as discussed in Section 2. Semantic descriptors are a compact notation for describing all objects that can be found in the cache. It allows us to adjust cached query groups to incorporate new data from an incoming query so that no irrelevant data is cached along with the relevant ones, thus reducing overhead in managing the cache, to dynamically adapt the cache content based on the pattern of user queries rather than just caching static clusters of tuples, so that data that logically belongs together is described by the same semantic descriptor, and to minimize the cost of cache lookup due to the compact representation of the cache content based on semantic descriptors.

A semantic caching scheme must at a minimum handle the following three tasks: first, decide whether the answer for a query resides in the cache or not; second, extract the parts of the answer available in the cache from the cache; and third, construct remainder query to fetch the remaining data. These tasks must be tuned to work for the characteristics of the particular application [9].

The recursive processing involved when navigating through hierarchies in main memory is no longer appropriate when storing those hierarchies on the disk. In our prior work, we have found that the computation of such recursive unions of joins and divisions to navigate the hierarchical structure can efficiently be accomplished by organizing the hierarchy as a MinMax tree [23] that encodes special positioned attributes (See Figure 5). These encodings include the left and right extents e_1, e_2 and the level values L, as defined in Section 2. Given a MinMax tree encoding, the recursive processing can be transformed into a set of fast range queries. For example, consider Figure 5 which models a continuous MinMax tree. To access the objects in nodes 4 and 7, a simple range query 0.25,0.75, 3 can be used (See also Figure 7).

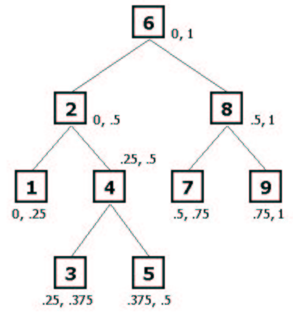

Figure 5: A MinMax tree.

MinMax tree [23] models the hierarchical exploration via our visual brush in XmdvTool [7] (see Section 2) as a two-dimensional exploration in which a selection window, called the *active window*, slides over an $n \times m$ grid of integers, called the *navigation grid*. n is the depth of the navigation tree and m is the number of leaf nodes in the tree. The objects (data clusters) now have a spatial representation that makes them selectable by the active window. As shown in [23, 24], this additional information, consisting of a level value L and two extents values e_1 and e_2 makes the objects behave like small rectangles $(e_1, e_2) \times L = (e_1, e_2, L)$ where $e_1 < e_2$, while still preserving their hierarchical structure (Fig. 6).

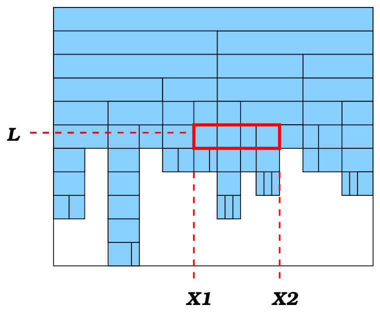

Figure 6: Objects as rectangles in XmdvTool and an active window $A = (x_1, x_2, L)$.

Objects are similar to active windows: they are both rectangular regions of the form $(e_1, e_2) \times L$. The cache content is described as a set of such range queries. The containment test of whether an object belongs to the active window or not reduces to an inclusion test between rectangles. For example, Fig. 7 presents such a range SQL query, assuming that our $N - dimensional$ cluster tree is stored in a table called $HIER(e_1, e_2, L, dim_1, dim_2, ..., dim_N)$, and $dim_1, dim_2, ..., dim_N$ denote the multidimensional values of the data points.

```
select   *
from     HIER
where    e_1 >= :x_1
and      e_2 <= :x_2
and      L = :L;
```

Figure 7: SQL Queries in XmdvTool for active window $A = (x_1, x_2, L)$.

4 Prefetching Strategies

In visualization applications such as ours, users typically spend a significant amount of time interpreting the graphical presentation of the selected data, while the processor and I/O system are idle. It is thus beneficial to predict what data the user will request next, and start fetching that data into the cache *before* the user asks for it. Thus, when the user requests that data later, she should perceive a faster response time. Due to the properties of visual exploration, such as the contiguity of queries, we can often accurately predict the user's next movement. Thus a viable solution to apply here is to *prefetch* the data before the request comes from the tool.

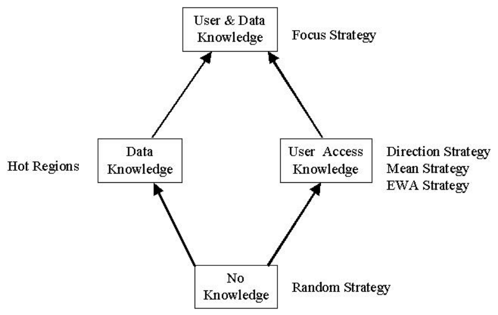

Figure 8: Hierarchy of Prefetching Strategies.

We have developed several strategies for prefetching, as described below, in order to perform a comparative evaluation of their applicability to exploratory visualization systems. Figure 8 organizes these prefetching strategies into a hierarchy based on different hints they utilize. The assumption is that the predictor can discover the hints *gradually* rather than complete knowledge *apriori*. The approach implies an evolutionary behavior; at the beginning, less information is available to the predictor and therefore the number of prefetching hints that it can discover (with a reasonable confidence) is also low. In time, more information (e.g., statistics) becomes available, and therefore more patterns (and implicitly hints) can be discovered. In all cases the prefetcher bases its strategy on the maximum amount of information it can find. In our case, we assume that the predictor can discover two types of navigation patterns. Specifically, we assume that the predictor can detect if the user tends to use more frequently the current navigation direction instead of changing it, and also it can detect if the data being analyzed has some regions of interest (so called *hot regions*) towards which the user will more likely go, sooner or later. Based on these assumptions, we have designed five prefetching strategies: *random* (S1), *direction* (S2), *focus* (S3), *mean* (S4), and *exponential weight average* (S5). No prefetching is referred to as S0.

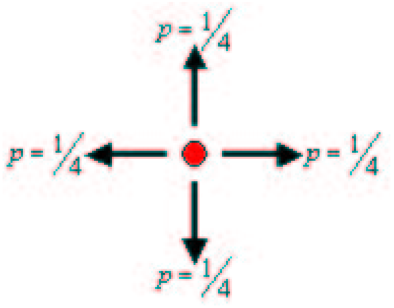

Figure 9: Random Strategy.

Random Strategy. As shown in Figure 9, strategy S1 (random) is based on randomly choosing the direction in which to prefetch next. The directions are either lateral (left or right at the same level in hierarchy) or vertical (increase or decrease level of detail). Our visualization tool only allows manipulation in either of those four directions. This strategy is appropriate when the predictor either cannot extract prefetching hints or provides hints with a low confidence measure.

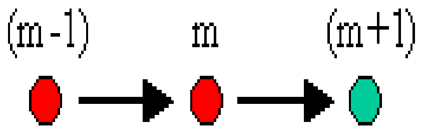

Figure 10: Direction Strategy.

Direction Strategy. Strategy S2 (direction) is analogous to the sequential prefetching scheme [3, 18]. This *direction* strategy assumes that the most likely direction of the next operation can be determined. It is intuitive, for instance, that the user will continue to use the same navigation tool for a while before changing to another one. In our system, each navigation tool of the structure-based brush happens to precisely control one direction only. Based on user's past explorations, the predictor would assign probabilities to the four directions. The prefetching strategy (S2) then is to "prefetch data in the direction" currently with the highest probability. As depicted in Figure 10, if $(m - 1)$ and m are the last two directions navigated into by the user, then the *direction* strategy may predict $(m + 1)$ as the next direction to be visited by the user in the same direction of the previous two movements.

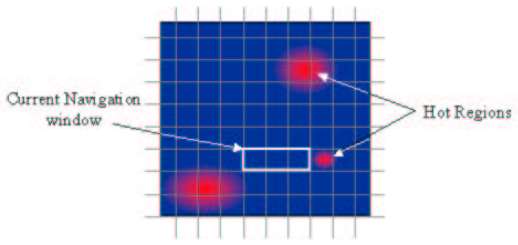

Figure 11: Hot Regions.

Focus Strategy. Strategy S3 (focus) uses information about the most probable next direction and hints about regions of high interest (hot regions) (Figure 11) in the data space as identified based on prior navigations of this same data by other users. We found the hot regions for each user by keeping the statistics of the regions visited by the user during the past explorations, and then maintaining regions that have frequency of visits above a particular threshold as hot regions. This strategy prefetches data in the given direction using the above mentioned direction heuristics. However, when a hot region is near the current navigation window, the prefetcher switches from the default *direction* prefetching to prefetch in that now more desirable direction. The hypothesis is that the user will likely stop at such a region of interest to explore those hot regions if she got close enough to notice them.

Vector Strategies. In these strategies, we use a three-dimensional vector to indicate the movement of the user's brushes - one for the start of brush e_1, one for the end of brush e_2, and one for the level L. For each user, we maintain a user trace containing the set of movement vectors over time, $m_1, m_2, \ldots, m_{n-1}$ where $m_i = (e_1^i, e_2^i, l_i)$. Each vector is calculated from the corresponding user's location and orientation, containing a move direction and a move distance. The general principle of such a vector strategy is to predict the $(n+1)$st movement vector, m_{n+1}, and prefetch objects that would be required if the user goes that way.

Given that we do not want to treat all movements of the user equally, but rather prefer to age older information over time, we propose to utilize two different strategies to predict the next location of the user: mean (S4) and exponential weighted average (S5), as depicted in Figures 12 and 13.

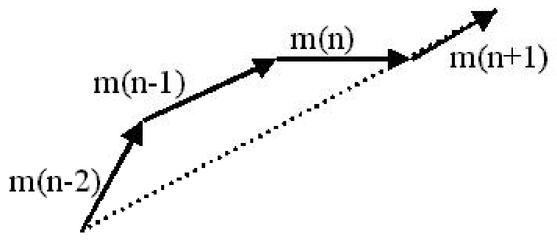

Figure 12: Mean Strategy.

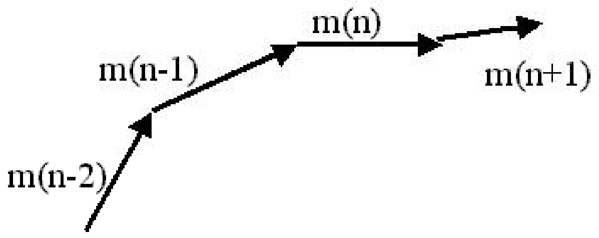

Figure 13: EWA Strategy.

In the mean strategy, the next movement vector is predicted to be the average of the previous n movement vectors m_i where $i = 1, 2, \ldots, n$. Our experiments have shown that the ideal value of n for our tool ranges from 2 to 4. The magnitude of the movement is determined by the average of the magnitudes of the previous movements. The predicted vector will then be:

$$m_{n+1} = \textstyle\sum_{i=1}^{n} m_i/n. \tag{1}$$

To adapt to changes in the user's moving patterns, the second vector strategy employs exponential smoothing. In this *Exponential Weight Average* strategy, we assign a weight to each previous movement vector m_i so that recent vectors have higher weights than movements from the distant past. We use an exponentially decreasing weight, α. The most recent vector receives a weight of 1; the previous vector a weight of α; the next previous one a weight of α^2, and so on.

The predicted vector hence is:

$$m_{n+1} = \textstyle\sum_{i=1}^{n} \alpha^{n-i} m_i / S_n \ \text{ with } \ S_n = \textstyle\sum_{i=1}^{n} \alpha^{n-i}. \tag{2}$$

Our experiments show that for both vector strategies, the number of history vectors we should consider as window is fairly small. Larger values of the window tend to lower the valid data being fetched.

5 Implementing Caching and Prefetching in XmdvTool

The caching and prefetching strategies described above have been implemented in XmdvTool 5.0 [26] (see Figure 14). XmdvTool is coded in C++ with Tcl/Tk and OpenGL primitives. The newly added modules are written in C with Pro*C (embedded SQL) primitives for Oracle8i. First, an off-line process clusters the flat data sets and then transforms the hierarchical data into MinMax trees [24], a pre-coded indexing structure that allows us to express hierarchical navigation as range queries. The transformed data is then loaded into the database.

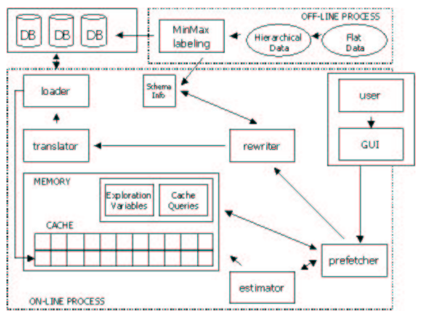

Figure 14: System architecture. Dotted-line rectangles show on-line versus off-line computation. Solid-line rectangles represent the modules. Ovals represent data. Arrows show control flow.

The interaction with the original system (shown as GUI in Figure 14) has been encapsulated as a database access API. We have implemented a prepare/iterate paradigm. When the user issues a new request, the front-end informs the back-end about the request by calling the *prepare* function. After that it can retrieve the desired objects one at a time from the buffer by repeatedly calling the *iterate* function.

Our semantic cache indexes the cache content with descriptors of the queries used to retrieve the cached data. This allows for a fast look-up, since only a few set-based operations are performed to compare a new query against cached queries (see Section 3). Details on our caching system can be found in [6].

When the system is idle, the *prefetcher* thread is created which communicates with the *estimator* to make decisions about the next most probable data to be prefetched depending on the prefetching strategy and the current cache content. The prefetching query is then passed to a *rewriter*. The *rewriter* consults the semantics of the cache (expressed by cached queries) and generates a set of sub-requests to adjust the data in the cache. Each sub-request is transformed into an SQL query by the *translator*. The queries are passed to the *loader*, which fetches the necessary objects from the database using a cursor and places them in the cache. Whenever the cache is full, the *estimator* removes the objects from the cache by examining probability values that depend on their semantic distance from the active region. On explicit user request, the fetching process is started by the *main* thread. If the *prefetcher* thread is still running, it is preempted and the contents of the cache are adjusted for consistency.

6 Experimental Evaluation

6.1 Experimental Setup

Due to space limitations, we only present a selected subset of our results below [6]. All experiments were conducted on an Alpha v4.0 878 DEC station, running Oracle 8i. We used C and embedded SQL for accessing Oracle 8i.

To eliminate data-specific effects, we used real and synthetic data sets. The real data sets were gathered from online repositories, while the synthetic data was generated using a random number generator. The experimental results reported below were based on a synthetic data set with 16,384 leaf nodes. We used real as well as synthetic user traces with different navigation patterns, each with around 300-3000 user requests.

Measures calculated from each navigation session include: number of objects displayed, object-based hit ratio, and latency. Since older display requests are cancelled if two requests come from the same user, loss of information (previous data not being displayed) is caused when the display requests are close in time to one another. The *number of objects* actually being displayed is a measure of the *visual quality*; a high value means more requested data is being displayed to the user. The *object-based hit ratio* is the number of objects already in cache over the total number of objects requested from the database. The *latency* is the total time that the user waited for her requests to be served. It also includes the time period when queries were cancelled due to new user requests. A high value means a low response time.

6.2 Experiments with Caching

Figure 15 shows the improvement in the performance of the system when the caching is turned ON or OFF on both the server (Oracle) and client side (our customized cache) respectively. Oracle optimizer was provided with cache hints “cache” and “nocache”.

Latency reduces by 85 percent just by caching at the client side as the hit ratio when client-side caching is turned ON is around 85 percent. Latency increases slightly with server-side caching turned ON as well as it caches the old data that the client may already have cached.

6.3 Varying Navigation Patterns

We study the effect of differences in navigation patterns by (1) varying the number of hot regions (Figures 16), (2) erratic versus directional navigation patterns (Figure 17), and (3) delay between user requests (Figure 18).

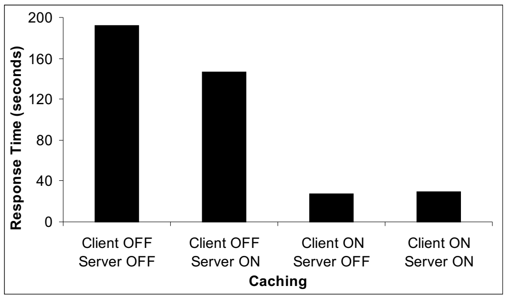

Figure 15: Latency vs. Caching.

Figure 16 shows the normalized latency for each strategy as the number of hot regions changes (on the X-axis), where:

$$NormalizedLatency = \frac{ActualLatency - MinLatency}{MaxLatency}$$

with $MinLatency$ and $MaxLatency$ the minimum and maximum value of latency from the observed latency values. Figure 16 confirms that any kind of prefetching improves the system response time (latency) compared to no prefetching at all. As the number of hot regions increases, the latency increases for all the prefetching strategies; but for the *focus* strategy, the latency reduces more in comparison with the other strategies. The *focus* strategy is the best as the number of hot regions increases.

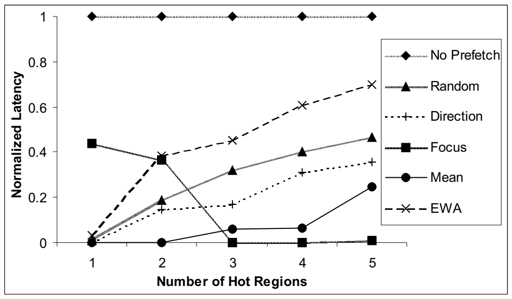

Figure 16: Latency vs. Number of Hot Regions (Normalized for Zero Latency).

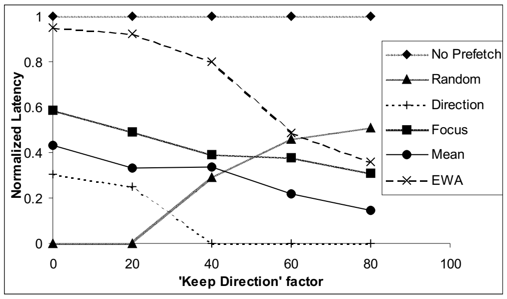

Figure 17: Latency vs. ‘Keep Direction’ factor.

Figure 17 shows the performance of each prefetching strategy when the navigation pattern changes from erratic to more directional. Recall that all strategies, except for the random strategy, use information about the direction of previous requests to make the next prediction. This figure confirms that the random strategy works best for erratic patterns, while the rest of the strategies show improved latency as the pattern becomes more directional and thus more predictable. The *direction* prefetching strategy has the best performance as the user becomes more 'directional'.

6.4 Effectiveness of the Prefetcher

Figure 18 charts the percentage of improvement (percent of latency reduced) in performance achieved by applying a prefetching strategy (*focus* strategy selected as representative) compared to not applying prefetching for various delays between user operations:

$$PercImprov = \frac{(LatencyWithoutP - LatencyWithP)}{LatencyWithoutP} * 100$$

.

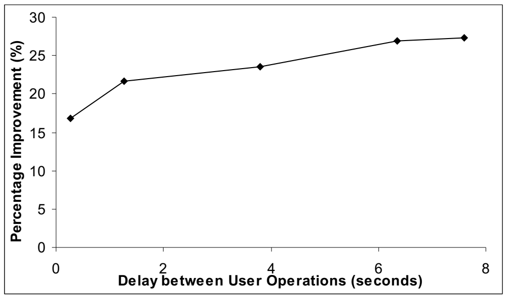

Figure 18: Prefetcher Effectiveness vs. Delay.

In Figure 18, the performance is improved up to 28%. This improvement is attained when the user manipulates the navigation interface (which means issues a new query to the backend) at the rate of one query per second. The curve flattens as the prefetching gets completed most of the time and is no longer preempted by user inputs. This indicates that an improvement in performance as the delay between user operations increases is not feasible beyond a certain point. This is as expected because the prefetching phase finishes most of the time and the system is idle.

Discussion. The focus strategy needs to analyze user traces (off-line) over a given data set to identify potential hot regions. This represents difficulty for its realization in some applications. The two vector strategies, *mean* and *ewa*, are less effective than the other prefetching strategies for our navigation environment since they try to fetch the data according to the vector-specific direction (which can be anywhere in the 360^o), while our navigation tool supports only four immediate directions. The *direction* strategy requires no prior knowledge and is simple, especially as its performance is nearly equal to that of the *focus* prefetching strategy. Based on this analysis, we have adopted the *direction* solution in our current XmdvTool system.

6.5 Experiments Using Real User Traces

We have performed a user-study with real users during which time our logging tool collected the traces of user explorations. The users were given different real datasets to visualize and find interesting patterns in the data. This allowed us to figure out the various ways users try to understand the data. These traces consisted of 30 minutes each for 20 different users. These traces when given as input to our tool under various system settings gave results similar to our synthetic user traces, confirming our conclusions outlined above. Due to space reasons actual charts are omitted here [6].

7 Related Work

Integrated visualization-database systems such as Tioga [1], IDEA [21] and DEVise [17] are most closely related to ours. Tioga [1] implements a multiple browser architecture for a *recipe*, a visual query. IDEA [21] is an integrated set of tools that supports interactive data analysis and exploration by providing multiple display views. In DEVise [17], a set of query and visualization primitives to support data analysis is provided. Special memory management techniques such as caching and prefetching have not been studied in the context of these systems.

Semantic caching is used for client-side caching and replacement in a client-server database system. It is aimed at providing support for navigational access to data, hence our proposal to apply them to visualization applications. We have implemented a hash-based caching structure inspired by [13, 4] to support efficient access to data. Most work on prefetching can broadly be classified into three classes: web prefetching [16], prefetching for memory caches by operating systems [19, 12], and I/O prefetching [15]. No work has been done to date focussed on prefetching for visualization applications. Web prefetching typically uses the idle time when the user is thinking what to do. We utilize the same principle. Similarly, the work done in I/O prefetching uses the I/O idle time to prefetch the data into the memory. The prefetching techniques in web prefetching [16] typically prefetch the pages most frequently visited by the user. This is similar to our *focus* strategy, in the sense of associating usage values with the object space instead of focussing on user trace analysis. *Mean* and *exponential weight average* strategies have been inspired by [2]. *Direction* strategy is similar to sequential prefetching strategies [3, 18].

8 Conclusions

To achieve scalability of exploratory analysis systems such as XmdvTool, good memory management strategies must be employed to reduce the overhead of I/O intensive database accesses. Our research identifies different properties of visualization environments exploitable for optimizing data access performance, applies semantic caching to visualization applications such as XmdvTool, develops prefetching techniques in support of visual exploration, implements both semantic caching and prefetching in XmdvTool, and performs experimental studies evaluating the proposed strategies in our XmdvTool system. Our experiments have shown that caching and prefetching at the client-side improves the performance of our visual environment considerably, for example, in some cases from approximately 190 seconds to 30 seconds. We note that these prefetching strategies are general and thus applicable to any applications that exhibit characteristics similar to those identified in Section 2.

Acknowledgements. We thank Jing Yang, Geraldine Rosario and other students in XMDV team for their XMDV tool development efforts - without them this work would not have been possible. We also thank students in the DSRG research group for helpful feedback on the work. Last but not the least, we would like to thank all the students who participated in our user study to help us get real user traces.

References

[1] A. Aiken, J. Chen, M. Lin, and M. Spalding. The Tioga-2 database visualization environment. *Lecture Notes in Computer Science*, 1183:181–190, 1996.

[2] J. Chim et al. On caching and prefetching of virtual objects in distributed virtual environments. *ACM Multimedia, pp.171–180*, Sept. 1998.

[3] F. Dahlgren, M. Dubois, and P. Stenström. Fixed and Adaptive Sequential Prefetching in Shared Memory Multiprocessors. In *Int. Conf. on Parallel Proc., 1993*, pages 56–63.

[4] S. Dar, M. J. Franklin, B. T. Jónsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. *VLDB, pp.330–341*, Sept 1996.

[5] D. DeWitt, D. Mayer, P. Futtersack, and F. Velez. A study of three alternative workstation-server architectures for object-oriented database systems. *VLDB'90*, pages 107–121. 1990.

[6] P. R. Doshi, E. A. Rundensteiner, and M. O. Ward. Prefetching for visual data exploration. Technical Report TR-02-07, WPI, CS Dept, 2002.

[7] Y. Fua, M. Ward, and E. Rundensteiner. Structure-based brushes: A mechanism for navigating hierarchically organized data and information spaces. *IEEE Trans. on Visualization and Computer Graphics, pp. 150-159*, 2000.

[8] Y. H. Fua, M. O. Ward, and E. A. Rundensteiner. Hierarchical parallel coordinates for exploration of large datasets. *IEEE Proc. of Visualization*, pages 43–50, Oct. 1999.

[9] P. Godfrey and J. Gryz. Answering queries by semantic caches. In *Proc of Database and Expert Systems Applications, Florence, Italy*, pages 485–498, Sept. 1999.

[10] S. Hibino and E. A. Rundensteiner. Processing incremental multidimensional range queries in a direct manipulation visual query. *ICDE, Florida, USA*, pages 458–465, 1998.

[11] S. Hibino and E. Rundersteiner. User interface evaluation of a direct manipulation temporal visual query language. *MULTIMEDIA*, pages 99–108, Nov. 1998. ACM Press.

[12] D. Joseph and D. Grunwald. Prefetching using Markov predictors. In *Intl Symposium on Computer Architecture (ISCA)*, pages 252–263, 1997.

[13] A. M. Keller and J. Basu. A predicate-based caching scheme for client-server database architectures. *VLDB Journal*, 5(1):35–47, 1996.

[14] A. Ki and A. E. Knowles. Adaptive data prefetching using cache information. In *International Conference on Supercomputing*, pages 204–212, 1997.

[15] P. Krishnan and J. Vitter. Optimal prediction for prefetching in the worst case. *ACM-SODA*, pages 392–401, 1994.

[16] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. *USENIX Symposium on Internet Technologies and Systems (ITS-97)*, pages 13–22, 1997.

[17] M. Livny, R. Ramakrishnan et. al. DEVise: Integrated querying and visualization of large datasets. *ACM SIGMOD, pp.301–312*, May 13-15, 1997.

[18] S. Manoharan and C. R. Yavasani. Experiments with sequential prefetching. *Lecture Notes in Computer Science*, 2110:322–331, 2001.

[19] R. H. Patterson et al. Informed prefetching and caching. *ACM Symposium on Operating Systems Principles*, pages 79–95, 1995.

[20] E. A. Rundensteiner, M. O. Ward, J. Yang, and P. R. Doshi. XmdvTool: Visual interactive data exploration and trend discovery of high-dimensional data sets. *Proceedings of ACM SIGMOD 2002*, page 631, 2002.

[21] P. G. Selfridge, D. Srivastava, and L. O. Wilson. Idea: Interactive data exploration and analysis. *ACM SIGMOD, pp.24–34*, June 1996.

[22] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Publishing, third edition, 1997.

[23] I. D. Stroe, E. A. Rundensteiner, and M. O. Ward. Minmax trees: Efficient relational operation support for hierarchy data exploration. Tech. Rep. TR-99-37, WPI, 1999.

[24] I. D. Stroe, E. A. Rundensteiner, and M. O. Ward. Scalable visual hierarchy exploration. In *Database and Expert Systems Appl.*, pages 784–793, Sept. 2000.

[25] M. O. Ward, J. Yang, and E. A. Rundensteiner. Hierarchical exploration of large multivariate data sets. *Proceedings Dagstuhl '00: Scientific Visualization*, May 2001.

[26] Xmdvtool home page. http://davis.wpi.edu/˜xmdv.

[27] J. Yang, M. O. Ward, and E. A. Rundensteiner. Interactive hierarchical displays: A general framework for visualization and exploration of large multivariate data sets. *Computers and Graphics journal*, 2003, (to appear).

[28] J. Yang, M. O. Ward, and E. A. Rundensteiner. An interactive tool for visually navigating and manipulating hierarchical structures. In *InfoVis* 2002, pp. 77-84.